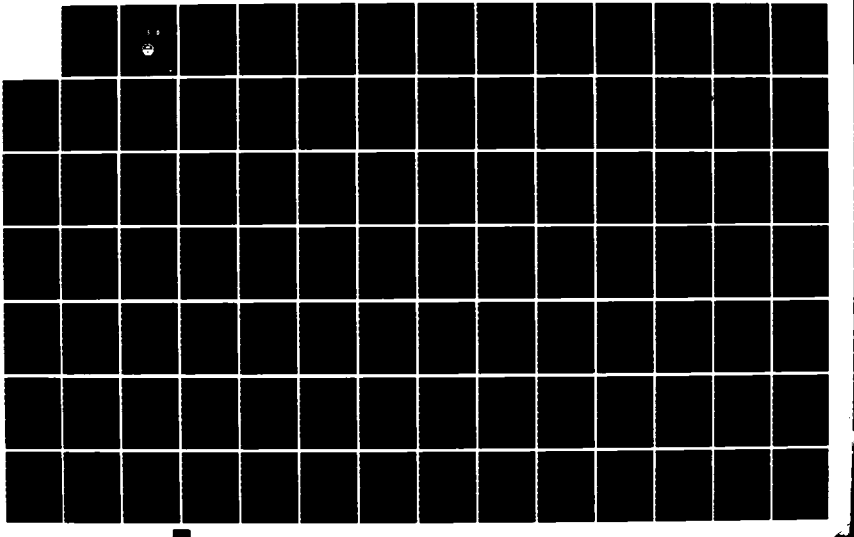
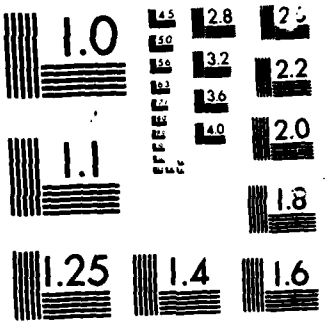
AD-A165 919 PROJECT STEAMER II USER'S MANUAL FOR THE STEAMER INTERACTIVE GRAPHICS PACKAGE(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA L STEAD AUG 81 NPRDC-TN-81-22 1/2

UNCLASSIFIED N00123-81-D-0794 F/G 5/9 NL





F630655 (1)

NPRDC TN 81-22 AUGUST 1981

PROJECT STEAMER: II. USER'S MANUAL FOR THE STEAMER INTERACTIVE GRAPHICS PACKAGE

NAVY PERSONNEL RESEARCH
AND
DEVELOPMENT CENTER
San Diego, California 92152

NPRDC Technical Note 81-22 August 1981

PROJECT STEAMER: II. USER'S MANUAL FOR THE STEAMER INTERACTIVE GRAPHICS PACKAGE

Larry Stead

Bolt Beranek and Newman, Inc.
Cambridge, MA 02138

Reviewed by
John D. Ford, Jr.

Released by
James F. Kelly, Jr.
Commanding Officer

Navy Personnel Research and Development Center
San Diego, California 92152

FOREWORD

This research and development was conducted under contract #N00123-81-D0794 with Bolt Beranek and Newman, Inc. in support of Navy Decision Coordinating Paper Z1177-PN (Advanced Computer-Aided Instruction), subproject Z1177-PN.03 (STEAMER: Advanced Computer-Based Training for Propulsion and Problem Solving). It was sponsored by the Chief of Naval Operations (OP-01). The main objective of the STEAMER effort is to develop and evaluate advanced knowledge-based techniques for use in low-cost portable training systems. The project is focused on propulsion engineering as a domain in which to investigate these computer-based training techniques.

This report, the second in a series on the STEAMER project, documents the general graphics software used for presenting diagrams and accepting input from users of the system. The first report (NPRDC TN 81-21) provided a taxonomy for generating explanations of how to operate complex physical devices. Intended users of these reports are system maintainers and other research personnel.

The contracting officer's technical representative was Dr. James Hollan.

JAMES F. KELLY, JR.
Commanding Officer

JAMES J. REGAN
Technical Director

SUMMARY

Problem

The main objective of the STEAMER effort is to develop and evaluate advanced knowledge-based techniques for use in low cost portable training systems. The project is focused on propulsion engineering as a domain in which to investigate these computer-based training techniques. The basic design of the STEAMER system includes a mathematical model of a propulsion plant that students and instructors inspect and control using diagrams presented on a color graphics display.

Objective

The objective of this effort was to document the general software used to present, create, and display diagrams and accept input from users of STEAMER. This documentation is intended as a reference manual for those who write STEAMER software.

Software Documentation

The STEAMER hardware consists of a color graphics terminal, a text terminal, and several graphical input devices. The graphics output software is organized in terms of four levels: (1) a level that mirrors precisely the graphics terminal functionality, (2) a level that provides user-defined coordinate mappings, (3) a level modelled after LOGO graphics that preserves the state of an abstract object called a "turtle," and (4) the topmost level, which includes a library of graphics objects such as pumps, graphs, gauges, pipes, and valves for constructing diagrams. The graphics input software allows the programmer to define abstract input devices and mappings between the input device and the screen.

Applications

Organizing the system in this way has permitted the building of a graphics editor to manipulate graphics objects so that diagrams can be rapidly created. The abstract input device concept allows easy experimentation with different interactive input devices. No software modifications are needed for the user to change from using a transparent touch panel to using the terminal joystick or even a keyboard.

CONTENTS

	Page
INTRODUCTION	1
Pr	1
	1
	1
The NED Driver	1
The Graphics Lib	2
The Turtle Graph nterface	3
Graphical Input	4
APPENDIX--DOCUM NTATION OF STEAMER SOFTWARE FUNCTIONS	A-0

INTRODUCTION

Problem

The main objective of the STEAMER effort is to develop and evaluate advanced knowledge-based techniques for use in low cost portable training systems. The project is focused on propulsion engineering as a domain in which to investigate these computer-based training techniques. The basic design of the STEAMER system includes a mathematical model of a propulsion plant that students and instructors inspect and control using diagrams presented on a color graphics display.

Objective

The objective of this effort was to document the general software used to present, create, and display diagrams and accept input from users of STEAMER. This documentation is intended as a reference manual for those who write STEAMER software.

RESULTS

Four major packages of software are described in this section. These are the AED driver, the graphics library, turtle graphics, and the graphical input package.

The AED driver provides a set of functions that enable opening and closing I/O, writing pixel values, manipulating masks, and manipulating color look-up tables. The graphics library provides a set of function-supporting general coordinate system transformations and additional utilities. The turtle graphics, modelled after the LOGO turtle, provides the highest level of available functionality by supporting an abstract "turtle" that maintains a state defined by its location, heading, and pencolor. Finally, the graphical input package provides a uniform interface over a variety of devices. Two types of input are supported, button and locator, in a fashion roughly analogous to the SIGGRAPH Core Graphics System.

The AED Driver

This section describes the AED driver calls that are concerned with opening and closing I/O, pixel writing values, masks, and color look-up table manipulation. All arithmetic is in fixed point.

The AED video output is a two-dimensional array of picture elements, or pixels. Pixels are the quanta of the image, the smallest portion of image area that the AED can control. Reflecting this, the frame buffer memory is a two-dimensional array, with eight bits in each array element representing a pixel.

The color of a pixel in the image is determined by using the eight-bit pixel value (pixval) as an index into a "color table." A color table consists of 256 "slots," one slot for each possible pixval. A slot contains a red, green, and blue (RGB) value that describes the color to be displayed.

The AED maintains three internal parameters that govern all frame buffer write operations, the write mask, the draw value, and the background value. (The current values of these parameters are stored in the free variables "aed:swm," "aed:sec," and "aed:sbc" respectively.) When a draw command (vector, circle, etc) is issued to the AED, the appropriate pixels are selected for updating. If the Nth bit of the write mask is on, then the Nth bit of each selected pixel will be replaced with the Nth bit of the draw value. If the write mask bit is off, the corresponding pixel bit is not changed. The background value is used in a similar fashion, but only during a screen clear (the "aed:clr" function).

The file "aed:dcls" contains compiler declarations for the AED functions. These functions are summarized below and documented in the appendix. The documentation includes a description of the function, the arguments it takes and the argument types, and any relevant notes about the function. Unless stated otherwise, functions always return a value of "t."

aed:black-ct
aed:close
aed:clr
aed:copy-ct
aed:create-ct
aed:date-time
aed:dump-ct
aed:get-cts-list
aed:gray-ct
aed:load-ct

aed:put-cts
aed:put-cts-list
aed:open
aed:reset
aed:rotate-cts-down
aed:rotate-cts-up
aed:sbc
aed:sbl
aed:sbl-off
aed:scc
aed:scp+
aed:scpx
aed:sct
aed:sctl
aed:sctl-list
aed:sec
aed:sls
aed:swm

The Graphics Library

The graphics library (GL) has several objectives:

1. Providing a level of functionality somewhat higher than the AED driver.

2. Supporting arbitrary coordinate systems in floating point, which also provides some measure of device independence.

3. Collecting miscellaneous graphics utilities.

4. Supporting graphics classes.

The GL package allows one to define an arbitrary coordinate system that is then mapped into the AED coordinates. Two-dimensional coordinate transformations, including scaling translation and rotation, are supported using the standard 3 x 3 matrix scheme. The matrices are Lisp objects that can be lambda-bound using a supplied macro. Subregions defined with respect to a coordinate system can be given their own local coordinate system so that all references within them are in convenient coordinates.

The "current address position" (CAP) is the position where the AED "pen" was last left. Many functions use the CAP as a starting point for efficiency purposes. Unless otherwise stated, function calls do not modify the CAP.

The file "gl:dcls" contains complier declarations for the GL functions. These functions are summarized below and documented in the appendix. The documentation

includes a description of the function, the arguments it takes and the argument types, and any relevant notes about the function. Unless stated otherwise, functions always return a value of "t."

gl:bfl
gl:bind-coordinates
gl:box
gl:boxrel
gl:circle
gl:dca
gl:ecu
gl:findy
gl:idiv$
gl:ifl
gl:in-region
gl:line
gl:linerel
gl:map-coordinates
gl:map-standard-coordinates
gl:mod$
gl:mov
gl:movrel
gl:point
gl:reflect-x
gl:reflect-y
gl:restore-coordinates
gl:rotate
gl:save-coordinates
gl:set-coordinates
gl:set-standard-coordinates
gl:solidbox
gl:solidboxrel
gl:solidcircle
gl:transform-coordinate
gl:writetext

By convention, four variables, "xl," "yb," "xr," and "yb" are used to describe rectangular regions, where "xl" is the x coordinate of the left side of the region, "yb" is the y coordinate of the bottom side, "xr" is the x coordinate of the right side, and "yt" is the y coordinate of the top side.

The Turtle Graphics Interface

This section describes the turtle graphics interface. Calls to turtle graphics functions may be freely intermixed with calls to the AED and GL packages.

All arithmetic is in floating point, and the turtle coordinate system ranges from (0.0, 0.0) at the lower-left corner to (1.0, 1.0) at the upper-right. No error checking is performed, so coordinates outside this range will cause trouble.

The state of the turtle is maintained in several free variables. These variables may be evaluated, SETQed, or lambda-bound at any time. In addition, each variable has a function of the same name that returns the variable's current value. The variables are listed below:

1. heading is the direction of the turtle in degrees. 0.0 degrees corresponds to the +X direction, 90.0 to +Y (i.e., heading increases counterclockwise).

2. pencolor is the color of the turtle pen. Pencolors currently supported are black, red, green, yellow, blue, magenta, cyan, and white.

3. penstate is "t" if the pen is down, "nil" if it is up.

4. screencolor is the color the screen will be cleared to by a "clearscreen" call. The pencolor should equal the screencolor to make the turtle "erase."

5. turtlestate is "t" if the turtle cursor is visible, "nil" if it is invisible.

6. xcoord is the turtle x position.

7. ycoord is the turtle y position.

The file "turtle:dcls" contains compiler declarations for the turtle functions. These functions are summarized below and documented in the appendix. The documentation includes a description of the function, the arguments it takes and the argument types, and any relevant notes about the function. Unless stated otherwise, functions always return a value of "t."

back
box
circle
clearscreen
deg-rad
fillregion
forward

heading
here
hideturtle
home
initturtle
killturtle
left
pencolor
pendown
penstate
penup
rad-deg
right
screencolor
setheading
setpencolor
setscreencolor
setturtle
setx
sety
setxy
showturtle
solidbox
solidcircle
turtlestate
xcoord
ycoord

Graphical Input

The graphical input package provides a uniform interface over a variety of devices. Two types of input are supported, button and locator, in a fashion roughly analogous to the SIGGRAPH Core Graphics System.

Locator devices, such as the Summagraphics tablet, or the Elographics touch screen, report a position specified by the user. The full range of the locator device is mapped onto the current GL coordinate system extents.

Button devices report a value representing the status of a button or set of buttons. Button values vary somewhat between the various physical devices. However, they all return "0" when "not depressed," so this fact may be used for simple predicate testing.

The file "gi:dcls" contains the compiler declarations for the graphical input functions. These functions are summarized below and documented in the appendix. The documen-

tation includes a description of the function, the arguments it takes and the argument types, and any relevant notes about the function. Unless stated otherwise, functions always return "t."

elo:button-read
elo:locator-read
gi:button-change
gi:button-clear
gi:button-read
gi:button-print
gi:button-trap
gi:locator-clear
gi:locator-latch
gi:locator-read
gi:locator-print
gi:locator-trap
gi:select-button
gi:select-device
gi:select-locator
gi:track
summa:button-read
summa:locator-read

APPENDIX

DOCUMENTATION OF STEAMER SOFTWARE FUNCTIONS

AED Driver A-1
Graphics Library A-29
Turtle Graphics Interface A-61
Graphical Input A-95

aed:black-ct aed:black-ct

Description

The aed:black-ct function sets all the RGB values in a color table to zero.

Usage

```
(declare (notype (aed:black-ct notype)))

(aed:black-ct ct)
```

where:

1. ct is a color table.
2. return value is ct.

Notes

The color table is not transmitted to the AED.

See the aed:create-ct function.

Description

The aed:close function closes AED I/O.

Usage

```
(declare (notype (aed:close)))

(aed:close)
```

where:

1. return value is "t" if AED I/O was closed, "nil" if AED I/O was already closed.

Description

The aed:clr function clears the frame buffer memory to the current background color.

Usage

```
(declare (notype (aed:clr)))

(aed:clr)
```

Notes

See the aed:sbc and aed:swm functions.

Description

The aed:copy-ct function copies contiguous sections of color table slots from one color table to another.

Usage

```
(declare (notype (aed:copy-ct notype fixnum notype fixnum)))

(aed:copy-ct ct-from start-from ct-to start-to count)
```

where:

1. ct-from is the source color table.
2. start-from is the first slot to be copied from the source color table.
3. ct-to is the target color table.
4. start-to is the first slot to be updated in the target color table.
5. count is the number of slots to be copied.

Notes

The same color table may be used as source and target.

Description

The aed:create-ct function creates a new color table, initialized to all zeros (black).

Usage

```
(declare (notype (aed:create-ct)))

(aed:create-ct)
```

where:

1. return value is the new color table.

Description

The aed:date-time function prints the current date and time in large letters on the AED. This is useful for dating video tapes.

Usage

```
(declare (notype (aed:date-time)))

(aed:date-time)
```

Description

The aed:dump-ct function writes a color table to a file.

Usage

```
(declare (notype (aed:dump-ct notype notype)))

(aed:dump-ct ct filespec)
```

where:

1. ct is the color table to be dumped.
2. filespec is the file to be written in Maclisp filespec format. A "mergef" is performed on filespec and "((* *) * ct)" to obtain the final filename.

Notes

See the aed:load-ct function.

Description

The aed:get-cts-list function returns the contents of a color slot as a list.

Usage

```
(declare (notype (aed:get-cts-list notype fixnum)))

(aed:get-cts-list ct slot)
```

where:

1. ct is a color table.
2. slot is the number of the slot.
3. return value is a list, "(R G B)".

Notes

See the aed:put-cts-list function.

Description

The aed:gray-ct function sets a color table to a 256 slot gray scale.

Usage

```
(declare (notype (aed:gray-ct notype)))

(aed:gray-ct ct)
```

where:

1. ct is a color table.
2. return value is ct.

Notes

See the aed:create-ct function.

Description

The aed:load-ct function loads a color table from a file.

Usage

```
(declare (notype (aed:load-ct notype notype)))

(aed:load-ct ct filespec)
```

where:

1. ct is an existing color table that the file will be read into.
2. filespec is the file to be read in Maclisp filespec format. A "mergef" is performed on filespec and "((* *) * ct)" to obtain the final filename.

Notes

See the aed:dump-ct function.

Description

The aed:put-cts function sets a color table slot.

Usage

```
(declare (notype (aed:put-cts notype fixnum fixnum fixnum fixnum)))

(aed:put-cts ct slot r g b)
```

where:

1. ct is a color table.
2. slot is the color table slot to be updated.
3. r is the new red value.
4. g is the new green value.
5. b is the new blue value.

Description

The aed:put-cts-list function sets a color table slot.

Usage

```
(declare (notype (aed:put-cts-list notype fixnum notype)))

(aed:put-cts-list ct slot list)
```

where:

1. ct is a color table.
2. slot is the slot number to be set.
3. list is a list, "(R G B)", of the new color slot values.

Notes

See the aed:get-cts-list function.

Description

The aed:open function will open AED I/O, only if it is not already open.

Usage

```
(declare (notype (aed:open)))

(aed:open)
```

where:

1. return value is "t" if AED I/O was opened, "nil" if AED I/O was already open.

Notes

The free variable "aed:open?" is "t" if the AED is open, otherwise it is "nil".

Description

The aed:reset function flushes all pending AED I/O, and initalizes the AED.

Usage

```
(declare (notype (aed:reset)))

(aed:reset)
```

Notes

Aed:reset takes approximately two seconds to complete.

Description

The aed:rotate-cts-down function rotates a contigious segment of color table slots down by one slot, with the bottom slot going to the top slot.

Usage

```
(declare (notype (aed:rotate-cts-down notype fixnum fixnum)))

(aed:rotate-cts-down ct start length)
```

where:

1. ct is a pointer a color table.
2. start is the first slot in the segment to be rotated.
3. length is the number of slots to be rotated.

Description

The aed:rotate-cts-up function rotates a contigious segment of color table slots up by one slot, with the top slot going to the bottom slot.

Usage

```
(declare (notype (aed:rotate-cts-up notype fixnum fixnum)))

(aed:rotate-cts-up ct start length)
```

where:

1. ct is a color table.
2. start is the first slot in the segment to be rotated.
3. length is the number of slots to be rotated.

Description

The aed:sbc function sets the value that the AED memory will be set to after a call to the "aed:clr" function.

Usage

(declare (notype (aed:sbc fixnum)))

(aed:sbc pixval)

where:

1. pixval is the pixel value.

Notes

See the aed:clr and aed:swm functions.

Description

The aed:sbl function sets up the AED such that a pixel value maps alternately thru the AED color table and a new set of RGB values, producing a blink effect.

Usage

```
(declare (notype (aed:sbl fixnum fixnum fixnum fixnum fixnum
fixnum)))

(aed:sbl pixval r g b current new)
```

where:

1. pixval is the pixel value to be blinked.
2. r is the new red map value.
3. g is the new green map value.
4. b is the new blue map value.
5. old is the number of fields that the current color table will be used to map pixval.
6. new is the number of fields that the RGB arguments will be used to map pixval.

Notes

Up to eight pixel values can be blinked simultaneously.

The AED color table is not changed by this function.

A field is 1/60th of a second.

See the "aed:sbl-off" function.

Description

The aed:sbl-off function stops blinking started by the "aed:sbl" function.

Usage

(declare (notype (aed:sbl-off fixnum)))

(aed:sbl-off pixval)

where:

1. pixval is the pixel value which should no longer blink.

Description

The aed:scc function sets the pixel value that the cursor will use.

Usage

```
(declare (notype (aed:scc fixnum fixnum fixnum)))

(aed:scc pv1 pv2 blink-time)
```

where:

1. pv1, pv2 are pixel values.
2. blink-time is in units of 1/60th of a second.

Notes

The cursor will use pv1 for for the interval specified by blink-time, then pv2, then pv1 again, etc. If blinking is not desired, pv1 and pv2 should be equal.

Description

The aed:scp+ function instructs the AED to use a "+" shaped cursor.

Usage

```
(declare (notype (aed:scp+)))

(aed:scp+)
```

Description

The aed:scpx function instructs the AED to use a "x" shaped cursor.

Usage

```
(declare (notype (aed:scpx)))

(aed:scpx)
```

Description

The aed:sct function transmits segments of a color table to the AED.

Usage

```
(declare (notype (aed:sct notype fixnum fixnum)))

(aed:sct ct start count)
```

where:

1. ct is a color table.
2. start is the first slot to be transmitted.
3. count is the number of slots to be transmitted.

Description

The aed:sctl function updates one slot of a color table, and transmitts the new slot to the AED.

Usage

(declare (notype (aed:sctl notype fixnum fixnum fixnum fixnum)))

(aed:sctl ct slot r g b)

where:

1. ct is a color table.
2. slot is the slot to be updated.
3. r is the red value.
4. g is the green value.
5. b is the blue value.

Description

The aed:sctl-list function updates one slot a color table and transmitts the new slot to the AED.

Usage

```
(declare (notype (aed:sctl-list notype fixnum notype)))

(aed:sctl-list ct slot list)
```

where:

1. ct is a color table.
2. slot is the slot to be updated.
3. list is a list, "(R G B)", of the new slot values.

Description

The aed:sec function sets the draw pixel value.

Usage

```
(declare (notype (aed:sec fixnum)))

(aed:sec pixval)
```

where:

1. pixval is the new draw value.

Notes

The pixel value actually written depends both on the pixval value and the write mask. See the introduction and the aed:swm function.

Description

The aed:sls function sets specifies a broken line template.

Usage

```
(declare (notype (aed:sls fixnum fixnum)))

(aed:sls template scale)
```

where:

1. template is a binary encoding of the line template, with "1" bits corresponding to visible portions of the line and "0" bits to invisible portions. Template = 255 will yield solid lines.

2. scale is a scale factor for the template, and can be 1, 2, 4, or 8.

Description

The aed:swm function sets the AED write mask, which governs all write operations into AED memory.

Usage

```
(declare (notype (aed:swm fixnum)))

(aed:swm mask)
```

where:

1. mask is the new write mask.

gl:bfl gl:bfl

Description

The gl:bfl function fills the interior of a closed region. The CAP is a point inside the region, and the region boundary is defined by pixels of the specified value.

Usage

```
(declare (notype (gl:bfl fixnum)))

(gl:bfl pixval)
```

where:

pixval is the value of the pixels that define the region boundary.

Notes

The CAP is unchanged.

Description

The gl:bind-coordinates macro lambda binds a new mapping matrix. Functions inside the macro body see the new matrix, and may modify it destructively. When the macro returns, the new matrix is popped, and the matrix which existed prior to the gl:bind-coordinates call is restored.

Usage

(gl:bind-coordinates form1 form2 ... formN) where:

formI is any lisp form.

1. return value is the value of formN.

Notes

See the gl:save-matrix and gl:restore-matrix functions.

Description

The gl:box function draws the perimeter of a rectangle. One corner of the rectangle is at the CAP, and the opposite corner is at a specified position.

Usage

```
(declare (notype (gl:box flonum flonum)))

(gl:box x y)
```

where:

1. x is the x coordinate of the corner opposite the CAP.
2. y is the y coordinate of the corner opposite the CAP.

Notes

The CAP is unchanged.

Description

The gl:boxrel function draws the perimeter of a rectangle. One corner of the rectangle is at the CAP, and the opposite corner is determined by the length of the rectangle's sides.

Usage

```
(declare (notype (gl:boxrel flonum flonum)))

(gl:boxrel dx dy)
```

where:

1. dx is the length of the horizontal side.
2. dy is the length of the vertical side.

Notes

The CAP is unchanged.

Dx and dy may be negative.

Description

The gl:circle function draws the perimeter of a circle. The center of the circle is the CAP.

Usage

(declare (notype (gl:circle flonum)))

(gl:circle radius)

where:

1. radius is the radius of the circle.

Notes

The CAP is unchanged.

Description

The gl:dca function sets the CAP at a specified position, and turns on the cursor there.

Usage

```
(declare (notype (gl:dca flonum flonum)))

(gl:dca x y)
```

where:

1. x is the x position of the cursor.
2. y is the y position of the cursor.

Notes

See the gl:ecu function.

Due to a feature/bug in the AED, the cursor must be turned off before calling any function that does drawing.

Description

The gl:ecu function turns the AED cursor off.

Usage

```
(declare (notype (gl:ecu)))

(gl:ecu)
```

where:

Notes

See the gl:dca function.

Description

The gl:findy function, given two points which define a line and an x value, computes y such that (x,y) is on the line.

Usage

```
(declare (notype (gl:findy flonum flonum flonum flonum flonum)))

(gl:findy xline yline xline2 yline2 x)
```

where:

1. (xline,yline) is a point on the line.
2. (xline2,yline2) is a second point on the line.
3. x is the x value of a point of the line.
4. return value is the y value corresponding to x.

Description

The gl:idiv$ function returns the "quotient" of a floating point divide.

Usage

```
(declare (flonum (gl:idiv$ flonum flonum)))

(gl:idiv$ dividend divsor)
```

where:

1. dividend is the dividend.
2. divsor is the divsor.
3. return value is the quotient.

Example

(gl:idiv$ 3.0 1.2) returns 2.0

Description

The gl:ifl function fills a closed region. The CAP is a point inside the region, and the boundary is defined by pixels whose values differ from the value of the pixel at the CAP.

Usage

```
(declare (gl:ifl))

(gl:ifl)
```

where:

Notes

Description

The gl:in-region function determines whether a point is inside a rectangular region.

Usage

```
(declare (notype (gl:in-region flonum flonum flonum flonum
flonum flonum)))

(gl:in-region xl yb xr yt xpt ypt)
```

where:

1. xl, yb, xr, yt defines the region.
2. xpt is the x coordinate of the point to be tested.
3. ypt is the y coordinate of the point to be tested.
4. return value is "t" if the point is inside the region, otherwise "nil".

Description

The gl:line function draws a line from the CAP to the position specified.

Usage

```
(declare (notype (gl:line flonum flonum)))

(gl:line x y)
```

where:

1. x is the x position of the second endpoint.
2. y is the y position of the second endpoint.

Notes

The new CAP is (x,y).

Description

The gl:linerel function draws a line from the CAP to CAP + (dx, dy).

Usage

```
(declare (notype (gl:linerel flonum flonum)))

(gl:linerel dx dy)
```

where:

1. dx is the length of the line in the x direction.
2. dy is the length of the line in the y direction.

Notes

The new CAP is the current CAP + (dx, dy).

Description

The gl:map-coordinates function modifies the mapping matrix such that a rectangular region may be referenced with a new coordinate system.

Usage

(declare (notype (gl:map-coordinates fixnum fixnum fixnum fixnum fixnum fixnum fixnum fixnum)))

(gl:map-coordinates cxl cyb cxr cyt nxl nyb nxr nyt)

where:

1. cxl cyb cxr cyt defines the region in the current coordinates.
2. nxl nyb nxr nyt defines the region in the new coordinates.

Notes

gl:map-standard-coordinates gl:map-standard-coordinates

Description

The gl:map-standard-coordinates function modifies the current mapping matrix such that a rectangular region may be referenced by x and y coordinates whose values range between 0.0 and 1.0.

Usage

(declare (notype (gl:map-standard-coordinates flonum flonum flonum flonum)))

(gl:map-standard-coordinates xl yb xr yt)

where:

1. xl yb xr yt define the rectangular region.

Notes

See the gl:map-coordinates function.

Description

The gl:mod$ function computes the "remainder" of a floating point division.

Usage

```
(declare (flonum (gl:mod$ flonum flonum)))

(gl:mod$ dividend divsor)
```

where:

1. dividend is the dividend.
2. divsor is the divsor.
3. return value is the remainder.

Example

(gl:mod$ 1.2 .8) returns .4

Description

The gl:mov function sets the CAP.

Usage

```
(declare (notype (gl:mov x y)))

(gl:mov x y)
```

where:

1. x is the new x position.
2. y is the new y position.

Description

The gl:movrel function sets the CAP relative to the the current one.

Usage

```
(declare (notype (gl:movrel flonum flonum)))

(gl:movrel dx dy)
```

where:

1. dx is the relative x distance from the current CAP to the new one.
2. dy is the relative y distance from the current CAP to th new one.

Description

The gl:point function draws a point at the CAP.

Usage

```
(declare (notype (gl:point)))

(gl:point)
```

Description

The gl:reflect-x functions modifies the current mapping matrix such that points are reflected about an x-axis.

Usage

```
(declare (notype (gl:reflect-x flonum)))

(gl:reflect-x x)
```

where:

1. x defines the x axis.

Notes

See the gl:reflect-y function.

Description

The gl:reflect-y function modifies the current mapping matrix such that points are reflected about a y axis.

Usage

```
(declare (notype (gl:reflect-y flonum)))

(gl:reflect-y y)
```

where:

1. y defines the y axis.

Notes

See the gl:reflect-x function.

Description

The gl:restore-coordinates function sets the current mapping matrix to the specified matrix.

Usage

```
(declare (notype (gl:restore-coordinates notype)))

(gl:restore-coordinates matrix)
```

where:

1. matrix is a mapping matrix, typically one returned from the gl:save-coordinates function.

Notes

See the gl:save-coordinates function.

Description

The gl:rotate function modifies the current mapping matrix such that points are rotated about a specified location.

Usage

```
(declare (notype (gl:rotate flonum flonum flonum)))

(gl:rotate x y angle)
```

where:

1. x,y is the point to rotate about.
2. angle is the amount of rotation in degrees.

Description

The gl:save-coordinates function returns a copy of the current mapping matrix.

Usage

```
(declare (notype (gl:save-coordinates)))

(gl:save-coordinates)
```

Notes

See the gl:restore-coordinates function.

Description

The gl:set-coordinates function sets the mapping matrix such that the screen is referenced by a specified coordinate system.

Usage

(declare (notype (gl:set-coordinates flonum flonum flonum flonum)))

(gl:set-coordinates xl yb xr yt)

where:

1. xl, yb, xr, yt defines the screen region.

Notes

See the gl:set-standard-coordinates function.

Description

The gl:set-standard-coordinates function sets the mapping matrix such that the screen is referenced by x and y values ranging between 0.0 and 1.0.

Usage

```
(declare (notype (gl:set-standard-coordinates)))

(gl:set-standard-coordinates)
```

Notes

See the gl:set-standard-coordinates function.

Description

The gl:solidbox function draws a filled in rectangle. One corner of the rectangle is at the CAP, and the opposite corner is at a specified position.

Usage

```
(declare (notype (gl:solidbox flonum flonum)))

(gl:solidbox x y)
```

where:

1. x is the x coordinate of the corner opposite the CAP.
2. y is the y coordinate of the corner opposite the CAP.

Notes

The CAP is unchanged.

Description

The gl:solidboxrel function draws the perimeter of a rectangle. One corner of the rectangle is at the CAP, and the opposite corner is determined by the length of the rectangle's sides.

Usage

```
(declare (notype (gl:solidboxrel flonum flonum)))

(gl:solidboxrel dx dy)
```

where:

1. dx is the length of the horizontal side.
2. dy is the length of the vertical side.

Notes

The CAP is unchanged.

Dx and dy may be negative.

Description

The gl:circle function draws a solid circle. The center of the circle is the CAP.

Usage

(declare (notype (gl:solidcircle flonum)))

(gl:solidcircle radius)

where:

1. radius is the radius of the circle.

Notes

The CAP is unchanged.

gl:transform-coordinate gl:transform-coordinate

Description

The gl:transform-coordinate function transforms a coordinate from one coordinate system to another.

Usage

(declare (notype (gl:transform-coordinate flonum flonum flonum flonum flonum)))

(gl:transform-coordinate old old2 new new2 old-coordinate)

where:

1. old maps into new.
2. old2 maps into new2.
3. old-coordinate is the value to be mapped into the new coordinate system.
4. return value is the mapped value of old-coord.

Description

The gl:writetext function writes a string of text at the CAP.

Usage

```
(declare (notype (gl:writetext notype notype)))

(gl:writetext text horizontal vertical)
```

where:

1. text is the string to be written.
2. horizontal controls the horizontal alignment of the text string relative to the CAP. The choices are "left" (text to the left of the CAP), "center", or "right".
3. vertical controls the vertical alignment of the text string relative to the CAP. The choices are "above" (text above CAP), "center", and "below".

Notes

back back

Description

The back function causes the turtle to move backwards on the current heading.

Usage

(declare (notype (back flonum)))

(back distance)

where:

1. distance is the distance to move backwards.

Description

The box function draws the perimeter of a rectangle.

Usage

```
(declare (notype (box flonum flonum)))

(box dx dy)
```

where:

1. dx is the length of the horizontal side.
2. dy is the length of the vertical side.

Notes

The current turtle heading and position are unchanged.

Dx and dy may be negative.

Description

The circle function draws the perimeter of a circle. The current turtle position is the center of the circle.

Usage

(declare (notype (circle flonum)))

(circle radius)

where:

1. radius is the radius of the circle.

Notes

The current turtle position and heading are unchanged.

Description

The clearscreen function clears the screen to the current screencolor, and homes the turtle.

Usage

(declare (notype (clearscreen)))

(clearscreen)

where:

Notes

See the home function.

Clearscreen sets the AED write mask to 255, so all eight memory planes will be cleared and write enabled.

Description

The deg-rad function converts degrees to radians.

Usage

```
(declare (flonum (deg-rad flonum)))

(deg-rad angle)
```

where:

1. angle is measured in degrees.
2. return value is the angle in radians.

Notes

See the rad-deg function.

Description

Fillregion floods the region containing the current turtle position with the current turtle color.

Usage

```
(declare (notype (fillregion)))

(fillregion)
```

where:

Notes

Current turtle position and heading are unchanged.

Description

The forward function causes the turtle to move forward on the current heading.

Usage

```
(declare (notype (forward flonum)))

(forward distance)
```

where:

1. distance is the distance to move forwards.

heading heading

Description

The heading function returns the value of the free variable "heading", which is the direction of the turtle in degrees.

Usage

```
(declare (flonum (heading)))

(heading)
```

where:

1. return value is the current heading in degrees.

here here

Description

The here function returns a list containing (in order) the current x position, y position, and heading.

Usage

(declare (notype (here)))

(here)

where:

1. return value is the list (x y heading).

Notes

See the setturtle function.

Description

The hideturtle function causes the turtle cursor to become invisible.

Usage

```
(declare (notype (hideturtle)))

(hideturtle)
```

where:

1. return value is the value of the free variable "turtlestate" just prior to the hideturtle call.

Notes

See the showturtle function.

Description

The home function puts the turtle at its initial position, the center of the screen. It is equivalent to "(setturtle '(0.5 0.5 0.0))".

Usage

```
(declare (notype (home)))

(home)
```

Description

The initturtle function initializes the turtle graphics system and should be called at the beginning of a session.

Usage

```
(declare (notype (initturtle)))

(initturtle)
```

Notes

Initturtle clears the screen to black, and sets the turtle state as follows:

```
(setq heading 0.0 pencolor 'white penstate t screencolor
'black turtlestate t xcoord .5 ycoord .5)
```

Description

The killturtle function cleans up the mess that turtles leave around and should be called at the end of a session.

Usage

```
(declare (notype (killturtle)))

(killturtle)
```

Notes

It only hurts for a second.

Description

The left function causes the turtle to rotate its heading counterclockwise.

Usage

```
(declare (notype (left flonum)))

(left turn)
```

where:

1. turn is the turn angle in degrees.

Description

The pencolor function returns the value of the free variable "pencolor".

Usage

```
(declare (notype (pencolor)))

(pencolor)
```

where:

1. return value is "pencolor".

Description

The pendown function changes the turtle state so that subsequent moves will leave a trace in the current pencolor.

Usage

```
(declare (notype (pendown)))

(pendown)
```

where:

1. return value is the value of the free variable "penstate" prior to the pendown call.

Description

The penstate function returns the value of the free variable "penstate".

Usage

(declare (notype (penstate)))

(penstate)

where:

1. return value is "penstate".

Description

The penup function changes the turtle state so that subsequent moves will not leave a trace in the current pencolor.

Usage

```
(declare (notype (penup)))

(penup)
```

where:

1. return value is the value of the "penstate" free variable prior to the penup call.

Description

The rad-deg function converts radians to degrees.

Usage

```
(declare (flonum (rad-deg flonum)))

(rad-deg angle)
```

where:

1. angle is measured in radians.
2. return value is the angle in degrees.

Notes

See the deg-rad function.

Description

The right function causes the turtle to rotate its heading clockwise.

Usage

```
(declare (notype (right flonum)))

(right turn)
```

where:

1. turn is the turn angle in degrees.

Description

The screencolor function returns the value of the free variable "screencolor".

Usage

```
(declare (notype (screencolor)))

(screencolor)
```

where:

1. return value is "screencolor".

Description

The setheading function sets the turtle direction.

Usage

```
(declare (notype (setheading flonum)))

(setheading direction)
```

where:

1. direction is the new direction in degrees.

Description

The setpencolor function sets the color the pen will draw with when down.

Usage

```
(declare (notype (setpencolor notype)))

(setpencolor color)
```

where:

1. color is an atom whose print name is the desired color.
2. return value is the pencolor just prior to the setpencolor call.

Notes

Pencolors currently supported are black, red, green, yellow, blue, magenta, cyan, and white.

Description

The setscreencolor function sets the color that the screen will be cleared to after a call to the "clearscreen" function.

Usage

```
(declare (notype (setscreencolor notype)))

(setscreencolor color)
```

where:

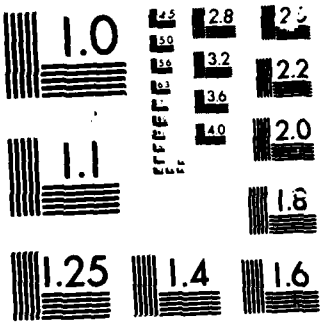
1. color is an atom whose print name is the desired color.
2. return value is the screencolor just prior to the setscreencolor call.

Notes

Screencolors currently supported are black, red, green, yellow, blue, magenta, cyan, and white.

AD-A163 919 PROJECT STEAMER II USER'S MANUAL FOR THE STEAMER INTERACTIVE GRAPHICS PACKAGE(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA L STEAD AUG 81 NPRDC-TN-81-22 2/2

UNCLASSIFIED N00123-81-D-0794 F/G 5/9 NL



Description

The setturtle function sets the turtle coordinates and heading.

Usage

(declare (notype (setturtle notype)))

(setturtle list)

where:

1. list is (x y heading).

Notes

See the here function.

If the pen is down, a trace will be left.

Description

The setx function sets the turtle x coordinate (the free variable "xcoord").

Usage

```
(declare (notype (setx flonum)))

(setx x)
```

where:

1. x is the new x.

Notes

If the pen is down, a trace will be left.

Description

The sety function sets the turtle y coordinate (the free variable "ycoord").

Usage

```
(declare (notype (sety flonum)))

(sety y)
```

where:

1. y is the new y.

Notes

If the pen is down, a trace will be left.

Description

The setxy function sets the turtle x and y coordinates (the free variables "xcoord" and "ycoord").

Usage

```
(declare (notype (setxy flonum flonum)))

(setxy x y)
```

where:

1. x is the new x.
2. y is the new y.

Notes

If the pen is down, a trace will be left.

Description

The showturtle function causes the turtle cursor to become visible.

Usage

```
(declare (notype (showturtle)))

(showturtle)
```

where:

1. return value is the value of the free variable "turtlestate" just prior to the showturtle call.

Notes

See the hideturtle function.

Description

The solidbox function draws a filled in rectangle.

Usage

(declare (notype (solidbox flonum flonum)))

(solidbox dx dy)

where:

1. dx is the length of the horizontal side.
2. dy is the length of the vertical side.

Notes

The current turtle position and heading are unchanged.

Dx and dy may be negative.

Description

The solidcircle function draws a filled in circle. The current turtle position is the center of the circle.

Usage

(declare (notype (solidcircle flonum)))

(solidcircle radius)

where:

1. radius is the radius of the solidcircle.

Note

Current turtle position and heading are unchanged.

Description

The turtlestate function returns the value of the free variable "turtlestate", which describes whether the turtle is up or down.

Usage

```
(declare (notype (turtlestate)))

(turtlestate)
```

where:

1. return value is "turtlestate".

Notes

Turtlestate is "t" if the pen is down, "nil" if the pen is up.

Description

Usage

```
(declare (notype (xcoord)))

(xcoord)
```

where:

1. return value is the turtle x position.

ycoord ycoord

Description

Usage

```
(declare (notype (ycoord)))

(ycoord)
```

where:

1. return value is the turtle y position.

elo:button-read elo:button-read

Description

The elo:button-read function divides the touch screen into four quadrants, and associates a button value with each.

Usage

(declare (notype (elo:button-read)))

(elo:button-read)

where:

1. return value is "t", if the Elographics is alive, "nil" otherwise.

2. *gi:button-val* is a free variable SETQed to "0" if the screen is not being touched, "1" if the upper right quadrant is being touched, "2" if the upper left quadrant is being touched, "4" if the lower left button is being touched, and "8" if the lower right quadrant is being touched.

Description

The elo:locator function reads the position of a finger on the surface of the Elographics.

Usage

```
(declare (notype (elo:locator)))

(elo:locator)
```

where:

1. return value is "t" if a finger is touching the surface, "nil" otherwise.
2. *gi:locator-x*, *gi:locator-y* are free variables SETQed to the x and y position of the finger.

Description

The gi:button-change function polls the current button device until it returns a value different from the one specified.

Usage

```
(declare (notype (gi:button-change fixnum)))

(gi:button-change oldval)
```

where:

1. oldval is the value that the button value must be different from.

Description

The gi:button-clear function clears the input buffer associated with the current button device.

Usage

```
(declare (notype (gi:button-clear)))

(gi:button-clear)
```

Description

The gi:button-print function is an infinite loop which prints the values returned by the gi:button-read function.

Usage

```
(declare (notype (gi:button-print notype)))

(gi:button-print clear-flag)
```

where:

1. clear-flag if non-nil, gi:button-clear will be called before gi:button-read each time in the loop.

Notes

Use cntl-G to stop.

Description

The gi:button-read function calls the current button device.

Usage

```
(declare (notype (gi:button-read)))

(gi:button-read)
```

where:

1. return value is the status returned by the current button function.
2. *gi:button-val* is a free variable which is SETQed by the current button function.

Notes

See the gi:select-button and gi:select-device functions.

Description

The gi:button-trap function polls the current button device until it returns the specified value.

Usage

```
(declare (notype (gi:button-trap fixnum)))

(gi:button-trap newval)
```

where:

1. newval is the value the current button device must return.

Description

The gi:locator-clear function clears the input buffer associated with the current locator device.

Usage

```
(declare (notype (gi:locator-clear)))

(gi:locator-clear)
```

Description

The gi:locator-latch function polls the locator and button devices alternatly until the sequence of button values "0", "non-zero", and "0" are reported. The most recent position of the locator device is available when the function returns.

Usage

```
(declare (notype (gi:locator-latch)))

(gi:locator-latch)
```

where:

1. *gi:locator-x*, gi:locator-y* are SETQed to the locator position.

Notes

See the gi:track function.

Description

The gi:locator-print function is an infinite loop which prints the values returned by gi:locator-read.

Usage

```
(declare (notype (gi:locator-print notype)))

(gi:locator-print clear-flag)
```

where:

1. clear-flag if non-nil, gi:locator-clear will be called before gi:locator-read each time in the loop.

Notes

Use cntl-G to stop.

Description

The gi:locator-read function calls the current locator device.

Usage

```
(declare (notype (gi:locator-read)))

(gi:locator-read)
```

where:

1. return value is the status returned by the locator function.
2. *gi:locator-x*, *gi:locator-y* are free variables SETQed by the locator function.

Notes

See the gi:select-locator and gi:select-device functions.

Description

The gi:locator-trap function waits for the locator device to return "t".

Usage

```
(declare (notype (gi:locator-trap)))

(gi:locator-trap)
```

Description

The gi:select-button function selects the function that will be used to read button values.

Usage

(declare (notype (gi:select-button notype)))

(gi:select-button device)

where:

1. device is an atom whose name is the device to be used.
2. device:button will be the function that gi:button-read calls.
3. return value is "t" if the select was successful, "nil" otherwise.

Notes

Currently supported button devices are "aed", "c100", "elo", "keyboard", and "summa".

Description

The gi:select-device function selects a device to act both as button and locator.

Usage

```
(declare (notype (gi:select-device notype)))

(gi:select-device device)
```

where:

1. device is an atom whose name is the device to be used.
2. return value is "t" if the select was sucessfull, "nil" otherwise.

Notes

Currently supported devices are "aed", "c100", "elo", "keyboard", and "summa".

See the gi:select-button and gi:select-locator functions.

Description

The gi:select-locator function selects the function that will be used to read locator values.

Usage

```
(declare (notype (gi:select-locator notype)))

(gi:select-locator device)
```

where:

1. device is an atom whose name is the device to be used.
2. device:locator is the function that gi:locator-read will call.
3. return value is "t" if the select was sucessful, "nil" otherwise.

Notes

Currently supported locator devices are "aed", "cl00", "elo", "keyboard", and "summa".

Description

The gi:track function will continuously display the position of the locator device by using the AED cursor. The function terminates (and the AED cursor is turned off) when the button device has reported a sequence of "0", "nonzero", and "0" values.

Usage

```
(declare (notype (gi:track)))

(gi:track)
```

where:

1. *gi:locator-x*, *gi:locator-y* are free variables SETQed to the last xy position the locator device reported before the function terminated.

Notes

See the gi:locator-latch function.

Description

The summa:button-read function reads the four buttons on top of the Summagraphics puck.

Usage

```
(declare (notype (summa:button-read)))

(summa:button-read)
```

where:

1. return value is "t", if the Summagraphics is alive, "nil" otherwise. (If the puck is not on the tablet surface, "nil" will be returned.)

2. *gi:button-val* is SETQed to "0" if no button is pressed, "1" if the yellow button is pressed, "2" if the white button is pressed, "4" if the blue button is pressed, and "8" if the green button is pressed.

Description

The summa:locator-read function reads the position of the puck on the tablet of the Summagraphics.

Usage

```
(declare (notype (summa:locator-read)))

(summa:locator-read)
```

where:

1. return value is "t", if the Summagraphics is alive, "nil" otherwise. (If the puck is not on the tablet surface, "nil" will be returned.)

2. *gi:locator-x*, *gi:locator-y* are free variables SETQed to the x and y position of the puck.

END
FILMED
4-86
DTIC